Finding Balanced Graph Bi-Partitions Using a Hybrid Genetic Algorithm

A.G. Steenbeek* E. Marchiori*,†,‡ A.E. Eiben†

Abstract— **In this paper we propose a hybrid genetic algorithm for the graph balanced bi-partition problem, a challenging NP-hard combinatorial optimization problem arising in many practical applications. The hybrid character of the GA lies in the application of a heuristic procedure to improve candidate solutions. The basic idea behind our heuristic is to identify and exploit clusters, i.e., subgraphs with a relatively high edge density. The resulting hybrid genetic algorithm turns out to be very effective, both in terms of quality of solutions and running time. On a large class of benchmark families of graphs, our hybrid genetic algorithm yields results of equal or better quality than those obtained by all other heuristic algorithms we are aware of, for comparable running times.**

I. Introduction

This paper introduces a hybrid genetic algorithm for finding approximate solutions of the graph balanced bi-partition problem (BP problem). The algorithm is a combination of a genetic algorithm and a local search procedure which is used for improving genetically created candidate solutions.

Given an undirected graph, the BP problem consists of dividing the set of its nodes into two disjoint subsets containing equal number of nodes[1] in such a way that the number of graph edges connecting nodes belonging to different subsets (i.e., the *cut size* of the partition) is minimized.

This combinatorial optimization problem arises in various practical applications like network partitioning, layout and floor planning [7], VLSI (very large-scale integration) circuit placement [19], etc. Due to its NP-hardness [11], the graph balanced bi-partition problem has been tackled by means of heuristic algorithms, which provide sub-optimal solutions of satisfactory quality in polynomial time. Various heuristic algorithms for the BP problem have been proposed (e.g., [13], [14]), also based on local search techniques like simulated annealing (e.g., [12]), tabu search (e.g., [8]) and hybrid genetic algorithms (e.g., [4], [16]). A recent summary of the approaches from the literature can be found in [9].

Most heuristics for the BP problem start from a bi-partition of the graph, and move nodes from one side to the opposite one according to a suitable criterion for reducing the cut size. This works well if the graph has a very regular structure. However, if this is not the case and there are 'dense' and 'sparse' subgraphs then one should try to exploit this structure. Our local search procedure is based on the notion of *clusters*. Intuitively, a cluster of a graph G is a subgraph with a density[2] which is significantly higher than the density of G. By the presence of many edges within a cluster it is likely that in a good bi-partition the whole cluster is on one side. Based on this idea, we propose a twofold extension of a genetic algorithm, consisting of:

- a procedure for finding clusters (pre-processing) and
- a procedure for good emplacement of clusters (local improvement).

The first procedure is used for determining the chromosomes representation, while the second procedure is used as operator on the chromosomes. We call the resulting hybrid algorithm 'cluster emplacement genetic algorithm' (CE-GA).

The procedure for finding clusters is executed before the (hybrid) GA is run. This phase can be seen as pre-processing with the purpose of identifying clusters and determining the genetic representation, where a chromosome represents a bi-partition of the set of clusters. Clusters are identified experimentally, by means of a traditional node swap heuristic (NSH) for the BP problem, which is run for a large number of times on independent input bi-partitions.

If the pre-processing phase indicates the absence of 'real' clusters, then the GA will work on nodes, which can be seen as clusters containing only one element. In this case NSH is also used as a local improvement procedure.

Otherwise, if the pre-processing phase indicates the presence of 'real' clusters, we use a novel 'cluster emplacement heuristic' (CEH). A (not necessarily balanced) bi-partition of the set of clusters is considered as input of CEH and then a number of iterations is performed. In each iteration we select a cluster from the side containing most nodes, and move it to the other side. The choice of a cluster is determined by the quality of the resulting bi-partition, both in terms of cut size and balance. A quasi-balanced bi-partition of the graph is obtained by iterating until no further improvement can be made.

We provide empirical evidence of the effectiveness of CE-GA by performing extensive experiments on two classes of random graphs, which are generally used as benchmark graphs for the graph balanced bi-partition problem.

The rest of the paper is organized as follows. In Section II we present the NSH algorithm, and in the following section we explain the pre-processing phase in more detail. Next, in Section IV we describe the CEH heuristic. In Section V

*CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
†Università Ca' Foscari di Venezia, via Torino 155, 30173 Mestre-Venezia, Italy
‡University of Leiden, P.O. Box 9512, 2300 RA Leiden, The Netherlands

[1] The graph is assumed to contain an even number of nodes; if it is not the case, a dummy node can be added.

[2] Recall that the density of a graph with N nodes and $|E|$ edges is $2 * |E|/(N * (N-1))$.

we introduce the CE-GA algorithm, and in Section VI we report on the results of our experiments and compare the performance of our algorithm with other ones. Finally, we conclude with a comparative discussion on our approach.

II. A node swap heuristic (NSH)

In the majority of the heuristic algorithms for the BP problem, an initial bi-partition of the graph is considered, and nodes are moved from one side to the other according to a suitable criterion for reducing the cut size (see e.g., [8]). A specific heuristic is determined by the choice of the nodes to be moved and by the way they are moved (e.g., simultaneous exchange of nodes, or sequential move from one to the opposite side).

In our approach, we use a simple node swap heuristic (NSH) for the BP problem based on the well known Kernighan-Lin algorithm [13].

Given an initial balanced bi-partition of the graph, a number of iterations are performed. Each iteration swaps a pair of nodes, selected from those pairs yielding the most decrease in the cut size. The iteration process continues until one can no longer find a pair of nodes for which a swap would reduce the cut size.

The running time per iteration is $O(d)$, where d is the maximum degree of a node. This is achieved by using a data structure similar to the one used in the Fiduccia-Mattheyses variant [10] of the Kernighan-Lin algorithm. Note that this data structure is created only once, before starting the iteration process, and it requires $O(|E|)$ running time, where E is the set of edges of the input graph.

The cut size is guaranteed to decrease at every iteration, therefore there is a maximum of $|E|$ iterations; however, in practice the number of iterations is much less. NSH is in itself very powerful, and can be used very effectively in combination with a genetic algorithm However, if the graph has a clustered structure then NSH may not perform satisfactorily.

III. Cluster identification

Recall, that a cluster of a graph G is a subgraph with a density which is significantly higher than the density of G. Therefore, it is very likely that in a clustered graph a minimal cut size can only be realized if clusters are not divided by the cut plane. Consequently, it is also likely that NSH keeps clusters on one side, because moving one node from a cluster to the other side would give a temporary increase in cut size. Besides, if the majority of the nodes of a cluster are in one side of the bi-partition, then moving one of the other cluster-nodes into that side is likely to give a reduction on the cut size.

We exploit this property in the pre-processing phase for the purpose of identifying the clusters of a graph. We apply NSH to the graph for a large number of times, each time starting with a randomly chosen initial bi-partition, and then we count how often each edge occurs in the cut plane (that is, its nodes occur on opposite sides) of the resulting bi-partitions. If an edge never appears in the cut plane then it is likely to belong to a cluster. This condition however is rather strong since the 'repair' property of NSH is not perfect. Therefore we decide to use a weaker condition which requires that an edge appears in the cut plane less than a small percentage of times, called *cluster-threshold*. We call such an edge a cluster-edge.

Clusters can now be defined as the components[3] of the graph obtained by temporarily removing all non-cluster-edges. Note that in this way every node of the graph belongs to exactly one cluster (which may consist of a single node), hence clusters are pairwise disjoint. The pre-processing phase applies NSH 100 times. Since NSH is very fast, in practice the pre-processing phase accounts only for a small part of the total computation time.

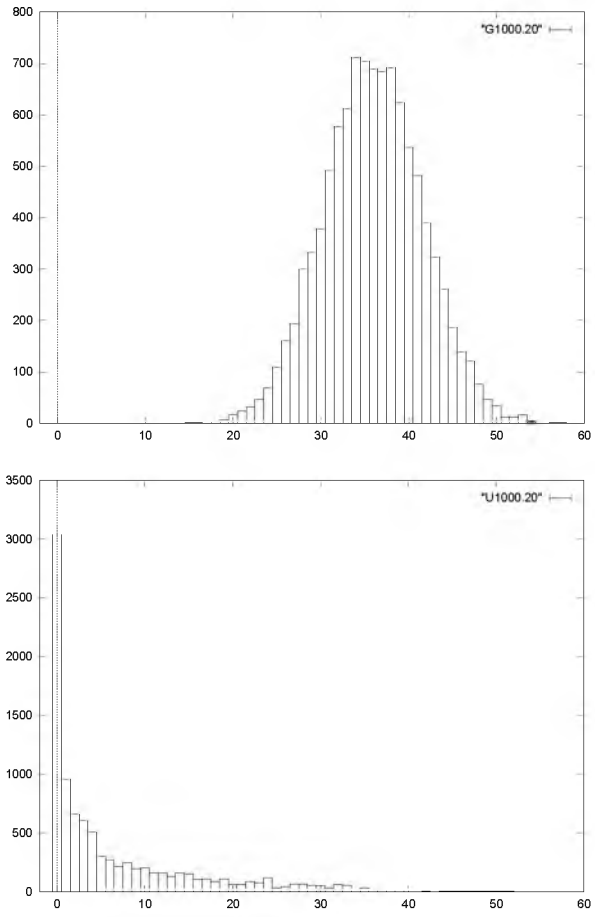

Fig. 1. Cut plane intersection frequency

Figure 1 illustrates the results of the pre-processing phase for two graphs of different types (defined in Section VI): a random graph $G1000.20$ and a geometric random graph $U1000.20$. NSH was run 100 times. For every n from 0 to 100, a bar is given, whose height corresponds to the number of edges that occur in the cut-plane n times out of 100. The results show a strongly clustered structure in $U1000.20$, while they do not indicate the presence of clusters for $G1000.20$.

The geometric graph $U1000.20$ consists of 1000 nodes and 9339 edges. After 100 NSH runs, we found that 3037

[3]A component of a graph is an inclusionwise maximal connected subgraph.

edges never crossed the cut plane. If we set the cluster-threshold at 3 percent, we get a total of 4650 cluster edges, resulting in 162 clusters of which 70 consist of only one node. The largest cluster contains 66 nodes, the size of the eighteen largest clusters adds up to 501 nodes.

In the case of the random graph $G1000.20$, we would have to set the cluster-threshold to 25 percent to get clusters of an acceptable size, this will however results in 'ghost clusters'.

In the current implementation, the threshold is automatically chosen. The program starts setting the threshold at 0 percent, and it increases its value if it does not detect enough clusters of a reasonable size; we do not allow a threshold of more than 10 percent. Note that relaxing the threshold (by increasing it) will result in the merging of clusters, thus producing larger clusters. More precisely, if $S(t)$ is the set of clusters found with cluster-threshold t, then , if $p > q$, every cluster in $S(p)$ is the union of one or more clusters in $S(q)$.

IV. Cluster Emplacement Heuristic

We have now all the instruments to define our novel heuristic algorithm for the BP problem, called 'cluster emplacement heuristic' (CEH). The CEH algorithm can be applied if the cluster identification algorithm indicates the presence of a clustered structure; the main purpose of CEH is to improve a bi-partition of the set of clusters.

A (not necessarily balanced) bi-partition of the set of clusters is considered as input, then a number of iterations is performed. Each iteration consists of the following steps:

1. Select a cluster (if possible);
2. If the cluster selection is not possible then stop; otherwise
   (a) Move the cluster to the opposite side;
   (b) Go to 1.

The selection of a cluster is clearly the core of the algorithm. A first requirement on cluster selection is that clusters from the smaller side are not allowed to be selected in case the bi-partition is unbalanced.

Furthermore, the selection procedure depends on the effect that a cluster move has on the cut size as well as on the degree of balance of the bi-partition. In order to formalize this conjunct effect, we introduce the notion of the 'move value' of the cluster.

Let $P = (A, B)$ be a bi-partition of the graph G, where A and B are the sets of nodes of the two sides. Denote by $csz(P)$ the cut size of P, and by $inb(P) = abs(|A| - |B|)$ the *inbalance* of the bi-partition, where $abs(\cdot)$ denotes the absolute value operator.

Now we define the evaluation function

$$E(P) = csz(P) + \alpha \cdot inb(P).$$

Here $\alpha > 0$ is a suitable constant, which affects the penalty given for the inbalance. Note that a lower value of $E(P)$ indicates a 'superior' partition, therefore if α has a high value then bi-partitions which are more balanced are preferred. A suitable value for α is chosen automatically based on the properties of the specific input graph. An alternative would be to slowly increase the value of α every iteration, however in the sequel we will assume α to be constant.

For a cluster C and a graph bi-partition P, denote by P' the bi-partition resulting by moving C to the opposite side. The *move value* of C is:

$$mov(C) = E(P) - E(P').$$

In this way the move value corresponds to the improvement made by moving cluster C to the other side. The selection procedure chooses a cluster having the highest move value amongst those with (strictly) positive move values.

The iteration stops when the move value of all the clusters are less than or equal to zero, which means we can not improve $E(P)$ by moving a cluster. The last (not necessarily balanced) bi-partition is the output of the CEH algorithm. It is clear that the algorithm will indeed always terminate, since every move reduces the value of $E(P)$, and a lower bound for $E(P)$ is the minimum cut size of the graph. An upper limit for the number of iterations is $\max(|E|, |E|/\alpha)$. In practice it is much less.

In the current implementation the running time per iteration is dominated by the time it takes to select a cluster. We examine every cluster in order to determine its move value. Computing the move value is very fast, because for every cluster we keep two counters. The first one holds the number of edges connecting a node from this cluster with a node from another cluster which is located on the same side of the partition. The other counter is similar, but in this case the other cluster has to be located on the other side of the partition. These counters have to be updated after every move (once per iteration); for efficiency every cluster contains a list of connected clusters along with the number of edges going to that cluster.

The CEH algorithm can be used to find a bi-partition of satisfactory quality, which may however be unbalanced; in that case in order to restore the balance one has to apply a node move algorithm to the resulting bi-partition.

A multistart version of the CEH algorithm used in this way yields results which are comparable to those of other powerful heuristic algorithms for the BP problem, if applied to clustered graphs. An alternative use of this heuristic is its incorporation into a genetic algorithm. We will show in the next section that the use of the CEH algorithm as a local improvement procedure on chromosomes results in a hybrid genetic algorithm with a very good performance.

V. A Hybrid Genetic Algorithm

Genetic algorithms have been shown to be rather effective when hybridized with non-genetic operators [17]. We introduce a hybrid genetic algorithm called Cluster Emplacement Genetic Algorithm (CE-GA) which applies the CEH algorithm to the chromosomes of the population in each iteration. Note that if the output of the pre-processing phase does not indicate the presence of clustered structures then we assume that the NSH algorithm is used instead of CEH as local improvement procedure in CE-GA. Observe that by seeing a node as a cluster consisting of that node

only, the NSH algorithm can be considered to act on clusters as well.

We use a generational GA that can be described as follows.

```
BEGIN
   t := 0;
   initialize P(t);
   apply Local Improvement to P(t);
   evaluate P(t);
   WHILE (NOT termination-condition) DO
      P(t+1) := { best chromosome of P(t) };
      FOR 2..poolsize DO
         select parent1 from P(t);
         select parent2 from P(t);
         child := cross_over( parent1, parent2 );
         apply Local Improvement to child;
         apply mutation to child;
         evaluate child;
         add child to P(t+1);
      ENDFOR
      t := t+1;
   ENDWHILE
END
```

Encoding: Since the heuristic acts on clusters, we consider a representation where a chromosome characterizes a bi-partition of the set of clusters of the graph. The length l of the chromosomes equals the number of clusters of the graph, hence l is at most equal to the number of nodes. The two sides of a bi-partition are denoted by 0 and 1 and a chromosome is a bit-string where the i-th bit b_i corresponds to cluster i: $b_i = 1$ (0) iff the cluster i is on the side 1 (0). Observe that the graph bi-partition described by a chromosome can be unbalanced.

Initialization and Population Size: The initial population consists of 50 chromosomes which are randomly generated. We prefer to start with chromosomes that represent a more or less balanced partition. For this purpose we use the following algorithm, where we assume that the clusters are ordered by size (number of nodes), such that $|C(i)| \geq |C(i+1)|$, where $|C(i)|$ denotes the number of nodes in cluster i.

```
BEGIN
  n  := number_of_nodes_in_the_graph;
  n0 := n/2;
  FOR i IN 1..number_of_clusters DO
     IF random() < n0/n THEN
        gene(i) := 0;
        n0 := n0 - |C(i)|;
     ELSE
        gene(i) := 1;
     ENDIF
  ENDFOR
END
```

The function *random()* is a uniform $[0, 1]$ random number generator. Moreover, n0 denotes the actual number of nodes which should be put in side 0 in order to have a balanced bi-partition. Note that if all clusters have size one, then this algorithm guarantees a balanced chromosome, and moreover every possible balanced bi-partition would have equal chance to be represented by this chromosome.

Fitness Function: Each chromosome *chrom* is evaluated by means of a fitness function F which is equal to the value $E(P)$ (see the previous section) of the corresponding graph bi-partition P:

$$F(chrom) = E(P)$$

The value of $E(P)$ is directly available as output of the CE-GA heuristic.

Selection: We use *tournament selection* [3] with tournament size 2: two parents are randomly chosen and the best of the two (i.e., the one with lower fitness) is selected. Moreover, we adopt *elitism* which copies the best chromosome of the actual population to the population of the next generation.

Crossover and Mutation: Since the application of the heuristic to similar chromosomes often yields identical chromosomes, the GA has a very strong tendency to converge prematurely. Therefore, in order to keep a high diversity among the elements of the population we use the following genetic operators. We apply a variation of the uniform crossover [20] which produces only one child. Crossover is always applied. Moreover, a mutation operator is used which flips all the bits of the chromosome; it is applied with probability 0.3. Observe that mutation does not affect the fitness of the chromosome to which it is applied.

Termination Condition: The algorithm will stop if it does not make an improvement in the last 100 generations. At this point the best found solution so far, is used as output of the GA. This solution may be (slightly) unbalanced, in this case it is processed by a simple node move algorithm that will restore the balance.

VI. Experimental Results

The CE-GA algorithm was implemented in C++ and it was run on a SGI-O2 workstation (with a 180 MHZ R5000 processor).

We have performed experiments on the following two classes of graphs, where a graph family in each class is determined by the parameters n and d:

- *Gn.d:* Random graphs with n nodes where we create an edge between any two nodes with probability p, this results in an expected node degree $d = p(n-1)$
- *Un.d:* Random geometric graphs with n nodes uniformly positioned in the unit square. An edge between two nodes is created if their Euclidean distance is less or equal than t. this gives an expected node degree $d = n\pi t^2$.

For every class, we consider a number of families obtained by choosing different values for the two parameters n and d, and we generate 100 graphs instances of each family (in total 3300 graphs). In order to compare our results, we consider the same graphs that have been used to test the performance of two heuristic algorithms: the EnTaS algorithm by Dell' Amico and Maffioli [8] and the Diff-Greedy algorithm by Battiti and Bertossi [1].

Graph family n d	CE-GA	EnTaS	Diff-G
G_250_2.5	25.0	26.1	26.1
G_250_5	114.3	117.6	121.5
G_250_10	342.3	345.5	351.8
G_250_20	862.7	860.0	869.3
G_250_40	1973.8	1975.9	1988.1
G_250_80	4337.3	4337.6	4354.3
G_500_2.5	49.9	54.9	53.6
G_500_5	228.2	237.5	242.8
G_500_10	681.6	685	702.9
G_500_20	1687.9	1699.4	1729.4
G_500_40	3876.8	3882.7	3927.4
G_500_80	8475.2	8495.2	8554.8
G_1000_2.5	101.3	116.6	110.8
G_1000_5	457.7	494.2	489.3
G_1000_10	1349.7	1386.5	1408.3
G_1000_20	3365.9	3394.6	3445.6
G_1000_40	7700.1	7724.9	7822.7
G_1000_80	16791.2	16837.2	16978.0
U_250_5	1.9	3.1	1.8
U_250_10	23.8	26.4	23.9
U_250_20	107.0	109.6	103.0
U_250_40	359.6	363.4	353.2
U_250_80	1081.3	1081.3	1108.5
U_500_5	2.2	4.8	2.2
U_500_10	31.3	39.7	32.0
U_500_20	140.8	143.8	144.3
U_500_40	472.9	474.9	502.3
U_500_80	1515.7	1516.2	1541.2
U_1000_5	4.6	6.9	2.3
U_1000_10	14.3	52.8	42.4
U_1000_20	189.9	195.0	198.3
U_1000_40	692.1	696.7	681.2
U_1000_80	2199.5	2202.5	2182.9

TABLE I

The EnTaS (Enhanced Tabu Search) algorithm is a very effective algorithm based on tabu search.

The Diff-Greedy algorithm is based on an heuristic which constructs a balanced bi-partition of small cut size starting from two empty sets and adding alternatively a node to each of the sets according with a suitable criterion for selecting that node. The heuristic is applied 1000 times and the best result found is considered as output of the algorithm.

Table I reports the results of our experiments with CE-GA together with the results obtained by EnTaS and by Diff-Greedy on the aforementioned random and geometric graphs. The first column specifies the considered graph family. In each family 100 graphs were generated and each algorithm was run on these 100 graphs once. Columns 2 through 4 contain the average cut size of the solution (for the GA, the best chromosome) at termination on the given graph family of the corresponding algorithms.

The performance of CE-GA on these graphs is very good (see second column). It outperforms EnTaS and Diff-Greedy on almost every family of random graphs ($Gn.d$). On the geometric graphs families ($Un.d$) CE-GA always outperforms EnTaS and it outperforms Diff-Greedy on more than half the cases. CE-GA continued until no improvements were found in the last 100 generations. This resulted in cpu-times ranging from 2 seconds for the small graphs up to 300 seconds for the largest ($G1000.80$) graphs. The results of the other two programs were taken from [8]. EnTaS always stopped after only 5 seconds cpu-time on a Pentium PC at 100 Mhz. Diff-Greedy would require cpu-times in the range 2 to 20 seconds on a machine with a performance comparable to the one we used. These differences in processing time makes it more difficult to compare the results, however allowing the other two programs the same amount of time, is known to give very little improvement in the quality of their solutions.

Graph	CE-GA		BFS-GBA		Diff-G	
	Avg	CPU	Avg	CPU	Res	CPU
G500.2.5	54.1	24.9	54.0	6.0	57	0.5
G500.05	221.8	23.7	222.1	8.1	231	1.0
G500.10	631.1	27.5	631.5	11.7	650	1.9
G500.20	1750.3	33.4	1752.5	21.6	1784	4.1
G1000.2.5	104.5	79.2	103.6	16.8	118	1.0
G1000.5	458.5	79.9	458.6	23.7	487	2.0
G1000.10	1374.6	79.5	1376.4	37.1	1435	4.1
G1000.20	3396.8	85.8	3401.7	62.3	3492	8.1
U500.5	2.2	13.4	3.7	7.5	2	1.0
U500.10	26.0	10.5	32.7	9.6	27	1.9
U500.20	178	26.3	179.6	11.5	179	3.6
U500.40	412.0	9.2	412.2	9.9	412	7.0
U1000.5	3.2	43.3	1.8	17.6	1	1.9
U1000.10	39	20.1	55.8	30.9	39	3.7
U1000.20	225.9	37.1	231.6	33.0	239	7.5
U1000.40	738.2	38.1	738.1	37.0	740	14.4

TABLE II

Next, we consider 16 specific graph instances from [12] of the two classes above introduced, which have been used in the past to test other heuristic algorithms (e.g., [2], [4], [12]). Here we report the results obtained using Diff-Greedy, and the results of a hybrid genetic algorithm (BFS-GBA) by Bui and Moon [4]. BFS-GBA uses a similar scheme, with a pre-processing phase for reordering the nodes of the graph, and a variation of the Kernighan-Lin heuristic as local improvement procedure. The authors employ a *steady-state* model ([21], [20]) on a population of 50 elements.

The table shows for every graph the average of the cut sizes of (the bi-partitions represented by) the best chromosomes on 1000 runs of BFS-GBA, and on 100 runs of CE-GA, as well as the result of Diff-Greedy. The reported cpu-time is the average time per run, except for Diff-Greedy where it is the (estimated) total time. Note that the CPU times reported in the table are not fully comparable since BFS-GBA was run on a Sun SPARC IPX and the Diff-Greedy timings were found using the formula $cptime = 1000 \cdot 0.0008 \cdot |E|$ contained in [1].

Table II indicates that the performance of CE-GA is very good. On the $Gn.d$ graphs, the results of CE-GA improve those of Diff-Greedy and are of the same quality as those of BFS-GBA. On the $Un.d$ graphs, the results of CE-GA are in most cases better than those of the other algorithms. In general, BFS-GBA is faster than CE-GA for graphs with lower density, and Diff-Greedy is always the fastest. However, one should realize that the results of Diff-Greedy hardly improve if one considers 10000 repetitions of the heuristic instead of 1000.

Note that the effect of the cluster identification phase depends on the class of graphs: for the majority of the random graphs $Gn.d$ there are no useful clusters found, hence the genetic algorithm uses the NSH heuristic for the

local improvement of the chromosomes. Nevertheless, for geometric random graphs *Un.d* the clustering phase turns out to be very effective. This is relevant because, as pointed out in [4], *Un.d* graphs are believed to be most similar to actual VLSI circuit and computer network graphs in the sense that they tend to have local 'clusters'.

VII. Conclusion

This paper introduced a hybrid genetic algorithm for the balanced bi-partition problem. In particular, we have investigated how the clustered structure of a graph can be exploited in a local improvement procedure. We have conducted extensive experiments on various families of graphs, and compared our method with other powerful heuristic algorithms, `EnTaS` based on local search, `Diff-Greedy` based on an effective construction procedure, and `BFS-GBA` based on genetic algorithms.

The performance of `CE-GA` is very good: the results are comparable or better than the best known results obtained using heuristic algorithms. This provides empirical evidence of the power of hybrid genetic algorithms for finding near optimal solutions of hard combinatorial optimization problems.

We would like to conclude with a discussion on related work. Many GA's for graph partitioning have been introduced ([4], [5], [6], [15], [18]). However, it is difficult to judge the performance of these works with respect to more popular heuristics, because very few experimental data are provided. Instead, we have used a large set of benchmark graphs, that allows one to compare our algorithm with the most recent and powerful heuristic algorithms for the `BP` problem.

The hybrid genetic algorithm `BFS-GBA` by Bui and Moon [4] we have considered in the experimental comparisons presents various similarities with `CE-GA`. This algorithm also uses a pre-processing phase, and a local improvement procedure. However, it acts on nodes, it uses a breadth first search on the input graph starting at a random node, and orders the nodes according to the order given by the adjacency matrix of the graph. Moreover, in the local improvement procedure, `BFS-GBA` swaps two sets of nodes from opposite sides, by means of a simple variation of the Kernighan-Lin heuristic. Instead, `CE-GA` uses the pre-processing for identifying clusters, and uses the novel heuristic `CEH` for swapping clusters in case the input graph presents a clustered structure. Concerning the GA features, `BFS-GBA` and `CE-GA` are based on different models (steady state and generational, respectively). However, they both try to maintain diversity in the population by introducing some disruption into the chromosomes. This is justified by the fact that both algorithms use a local improvement procedure and act on a small population (50 elements), thus diversity is needed in order to prevent premature convergence.

Acknowledgements

We would like to thank Roberto Battiti for pointing us to useful references, Mauro Dell' Amico for sending us the random instance generator used to perform the experiments on `EnTaS` and the solutions obtained.

References

[1] R. Battiti and A. Bertossi. Differential greedy for the 0-1 equicut problem. In D.Z. Du and P.M. Pardalos, editors, *Proc. DIMACS Workshop on Network Design: Connectivity and Facilites Location*, 1997. AMS, To appear.

[2] R. Battiti and A. Bertossi. Greedy and prohibition-Based heuristics for the graph partitioning. Technical report, University of Trento, Italy, PREPRINT UTM 512, February 1997.

[3] T. Blickle. Tournament selection. In T. Bäck, D. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*, pages C2.3:1–4. Institute of Physics Publishing Ltd, Bristol and Oxford University Press, New York, 1997.

[4] T.N. Bui and B.R. Moon. Genetic algorithm and graph partitioning. *IEEE Transactions on Computers*, 45(7):841–855, 1996.

[5] J.P. Cohoon, W.N. Martin, and D.S. Richards. A multi-population genetic algorithm for solving the k-partition problem. In R. Belew and L. Booker, editors, *Proc. 4-th International Conference on Genetic Algorithms*, pages 244–248. Morgan Kaufmann, 1991.

[6] R. Collins and D. Jefferson. Selection in massively parallel genetic algorithms. In R. Belew and L. Booker, editors, *Proc. 4-th International Conference on Genetic Algorithms*, pages 249–256. Morgan Kaufmann, 1991.

[7] W. Dai and E. Kuh. Simultaneous floor planning and global routing for hierarchical building block layout. *IEEE Transactions on CAD ICs Systems*, 1987.

[8] M. Dell'Amico and F. Maffioli. A new tabu search approach for the 0-1 equicut problem. In *Proc. Meta-Heuristics 1995: The State of Art*, pages 361–377, 1996.

[9] M. Dell'Amico and M. Trubian. Solutions of large weighted equicut problems. *European Journal on Operations Research*, 1997. To appear.

[10] C. Fiduccia and R. Mattheyses. A linear-time heuristic for improving network partitions. In *Proc. ACM/IEEE 19-th Design Automation Conference*, pages 175–181, 1982.

[11] M. Garey and D.S. Johnson. *Computers and Intractability: a Guide to the Theory of NP-completeness*. Freeman, San Francisco, 1979.

[12] D.S. Johnson, C.R. Aragon, L.A. McGeoch, and C. Schevon. Optimization by simulating annealing: An experimental evaluation; part I, graph partitioning. *Operations Research*, 37:865–892, 1989.

[13] B. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Systems Technical Journal*, 49:291–307, 1970.

[14] M. Laguna, T.A. Feo, and H.C. Elrod. A greedy randomized adaptive search procedure for the two-partitioning problem. *Operations Research*, 42(4):677–687, 1994.

[15] G. Laszewski. Intelligent structural operators for the k-way graph partitioning problem. In R. Belew and L. Booker, editors, *Proc. 4-th International Conference on Genetic Algorithms*, pages 45–52. Morgan Kaufmann, 1991.

[16] H. Pirkul and E. Rolland. New heuristic solution procedures for the uniform graph partitioning problem: Extensions and evaluation. *Computers and Operations Research*, 21(8):895–907, 1994.

[17] V.J. Rayward-Smith, I.H. Osman, C.R. Reeves, and G.D. Smith (eds.). *Modern Heuristic Search Methods*. Wiley, New York, 1996.

[18] Y. Saab and V. Rao. Stochastic evolution: A fast effective heuristic for some genetic layout problems. In *Proc. 27th ACM/IEEE Design Automation Conference*, 1990.

[19] W. Sun and C. Sechen. Efficient and effective placement for very large circuits. *IEEE Transactions on Computer-Aided Design*, 14(3):349–359, 1995.

[20] G. Syswerda. Uniform crossover in genetic algorithms. In J. Schaffer, editor, *Third International Conference on Genetic Algorithms*, pages 2–9. Morgan Kaufmann, 1989.

[21] D. Whitley. GENITOR: A different genetic algorithm. In *Proc. Rocky Mountain Conference on Artificial Intelligence*. Denver, 1988.